VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2018

Matej Klus

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Matej Klus**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: SCOVECO, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Konzultant bakalářské práce: Ing. Zdeněk Velart, Ph.D.

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem že túto bakalársku prácu som vypracoval samostatne. Uviedol som všetky literárne pramene a zdroje z ktorých som čerpal.

V Ostrave 23. apríla 2018

.....

podpis študenta

Súhlasím so zverejneným tejto bakalárskej práce podľa požiadaviek čl.26 odst. 9 Študijného a zkúškoveho poriadku pre štúdium v bakalárskych programoch VŠB-TU Ostrava.

V Ostrave 23. apríla 2018

.....

podpis zástupcu

Ďakujem firme SCOVECO s.r.o., že mi umožnili vykonávať individuálnu odbornú prax a tým získať znalosti pri práci na réalnom projekte. Ďalej ďakujem Ing. Zdeňkovi Velartovi, Ph.D., za riadenie a odboré rady pri absolvováni praxe. Taktiež svojmu vedúcemu Ing. Petrovi Lukášovi za pomoc a pripomienky k tejto práci. Za cennú podporu pri štúdiu ďakujem všetkým svojím blízkym.

Abstrakt

Táto bakalárska práca je zpracovaná na tému Absolvovania individuálnej odbornej praxi v spoločnosti SCOVECO s.r.o. V práci je definovaná charakteristika spoločnosti SCOVECO s.r.o. V ďalších častiach sú definované technológie, ktoré som využíval počas odbornej praxi , zadané úlohy a projekt na ktorom som pracoval, získané odborné skúsenosti počas absolvovania praxe , porovnanie a využite základných skúsenosti v praxi nadobudnutých počas štúdia na bakalárskom programe: Informatika a výpočetní technika (2612R025).

Kľúčové slová: SCOVECO s.r.o., Java, Liferay

Abstract

The topic of this Bachelor's thesis is based on individual professional practice in SCOVECO s.r.o (limited liability company). The thesis contains a definition of characteristics of the company. The other parts of the paper define technology used during my professional practice, assigned tasks, wrought project, gained professional experiences, comparison and usage of knowledge I get during my study in bachelor's program: Computer science and computer technology (2612R025).

Key Words: SCOVECO s.r.o., Java, Liferay

Obsah

Zoznam použitých skratiek a symbolov

API	–	Application Programmin Interface
ORM	–	Objektové relačné mapovanie
JVM	–	Java Virtual Machine
JSF	–	JavaServer Faces
SSO	–	Single sing-on
XML	–	eXtensible Markup Language
JSON	–	JavaScript Object Notation
s.r.o.	–	Spoločnosť s ručením obmedzeným
GET	–	Požiadavok na objekt s zasielaním dát
GIT	–	Distribuovaný systém riadenia revízií
IDE	–	Integrated Development Enviroment
XHTML	–	Rozšíriteľný hypertextový značkový jazyk
JPA	–	Java Persistance API

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Témou mojej bakalárskej práce bolo absolvovanie odbornej praxi vo firme SCOVECO s.ro. ako developer webovej aplikácie. Túto alternatívu vypracovania bakalárskej práce som si vybral predovšetkým z dôvodu získania praxe v informatickom obore. Chcel som sa naučiť niečo o práci v tíme, o pracovných postupoch a technológiách ktoré sa v súčasnosti využívajú pre tvorbu softvéru. Počas celej mojej praxe som spolupracoval na tvorbe webovej aplikácie s názvom Raynet Connector. Táto aplikácia slúži zákazníkom na import faktúr z ekonomického systému POHODA do systému Raynet CRM.

2 Zameranie firmy a pracovné zaradenie študenta

2.1 O firme

SCOVECO bola založená v roku 2011. Spoločnosť poskytuje vývoj na mieru. Vývoj informačných systémov, mobilných riešení alebo cloudového systému.

2.2 Pracovné zaradenie študenta a náplň práce

Počas mojej praxe som pracoval ako Java developer a počas toho som pracoval najmä na programovaní aplikácie Raynet Connector.

2.3 Pracovné prostredie

SCOVECO s.r.o má svoje kancelárie v podnikateľskom inkubátore VŠB-TUO v Ostrave, kde som počas absolvovania praxe pracoval. Komunikácia vo firme bola prevažne osobného charakteru.

3 Použité technológie a nástroje

K absolvovaniu praxe a k práci bolo nevyhnutné zoznámiť sa a naučiť sa pracovať s rôznymi technológiami.

3.1 Git

Git[2] je systém pre správu verzií, ktorú vytvoril Linus Torvalds v roku 2005 pre potrebu vývoja linuxového jadra. Tento systém je cielený na rýchlosť, podporu nelineárneho vývoja a schopnosť spravovať ako malé tak aj veľké projekty. Najmenšiou jednotkou systému je commit, teda revízia a ta by mala v ideálnom prípade obsahovať jednotlivú logickú časť práce.

3.2 Java

Java [5] je programovacia a počítačová platforma. Medzi jej najväčšie výhody bezprostredne patrí že programy vytvorené v nej sú multi-platformné. To znamená, že aplikácia napísaná v Jave môže byť spustená na ľubovoľnej platforme s nainštalovaným JVM (Java Virtual Machine). Tento jazyk ma široké použite od desktopových cez serverové až po mobilné aplikácie.

3.3 Hibernate

Hiberante[3] je framework napísaný v jazyku Java ktorý umožňuje tzv. objektovo-relačné mapovanie (ORM). Uľahčuje riešenie otázky zachovania dát objektov aj po ukončení behu aplikácie. Je jednou z implementácií Java Persistence API (JPA). Hibernate poskytuje spôsob, pomocou neho je možné zachovať stav objektov medzi dvoma spustenými aplikáciami. Udržuje dáta perzistentné. Dosahuje to pomocou ORM, čo znamená, že mapuje objetky v jazyku java na entity v relačnej databáze. K tomu používa tzv. mapovacie súbory, v ktorých je popísané, akým spôsobom sa majú dáta z objektu transformovať do databázy a naopak akým spôsobom sa z databázových tabuliek majú vytvoriť objekty.

3.4 JSF

JSF (Java Server Faces)[4] je v súčasnosti spôsob tvorby UI (User interface) webových aplikácií v Jave EE. JSF 2.x je hlavne v spojení s komponentnými frameworkami ako PrimeFaces, IceFace alebo RichFaces, je veľmi dobre použiteľná pre tvorbu moderných intranetových aplikácií.

Managed bean je základný druh Java triedy v JSF aplikáciách. Managed Bean vytvára základ pre dynamickosť JSF stránok. Obstaráva pre nich dáta a spravuje požiadavky, väčšinou typu GET alebo POST. Ide ju využiť aj na navigáciu pre webové aplikácie.

3.5 Java Eclipse

Eclipse[6] je integrované developerské prostredie (IDE) pre javu a ostatné programovacie jazyky ako napríklad C, C++, PHP a Ruby. Toto prostredie zahŕňa vývojové nástroje Eclipse Java(JDT) pre jazyk Java. Eclipse CDT pre C/C++ a Eclipse PDT pre PHP.

3.6 Apache Tomcat

Apache Tomcat® [10] je open source implementácia Java Servletov a JavaServer Pages, umožnújúca vytvárať dynamické stránky. Technológie aktívnych serverových stránok sú rozšírenou platformou. Apache Software Foundation je známa predovšetkým svojím open source web-serverom Apache. Stavia na využití najuniverzálnejšiej platformy pre programovanie jazyka Java. Apache Tomcat softvér je vyvinutý v otvorenom a participantovom prostredí a uvoľnení pod verziou Apache License 2.

3.7 PrimeFaces

PrimeFaces [8] je open source frontend framework pre JavaServer Faces, ktorý obsahuje viac ako 100 komponentov, na dotyk optimalizované mobilné karty, validáciu na strane klienta, tématický engine a mnoho ďalších funkcií. Jednotlivé komponenty sú vyvinuté s prihliadnutím na skrývanie zložitosti ale zachovanie flexibility.

3.8 Apache Shiro

Apache Shiro™ [9] je ľahko použiteľný systém zabezpečenia Java. Vykonáva autentifikáciu, autorizáciu, kryptografiu a správu relácií. Prostredníctvom rozhrania API spoločnosti Shiro je možné jednoducho a rýchlo zabezpečiť každú aplikáciu – od najmenších mobilných až po najväčšie webové a podnikové aplikácie.

3.9 GSON

Gson [1] je Java knižnica, ktorá sa používa na konverziu objektov Java na ich reprezentáciu JSON. Môže sa používať aj na konverziu reťazca JSON na ekvivalentný objekt Java. Gson môže pracovať s ľubovoľnými objektmi Java.

- Poskytuje jednoduché mechanizmy na konverziu Java objektov na JSON a naopak.
- Povoľuje vlastné reprezentácie objektov.
- Podporuje ľubovoľné zložité objekty.
- Vytvára kompaktný a čitateľný výstup JSON.

3.10 Liferay Portal

Liferay Portal [7] je open source podniková webová platforma pre vytváranie obchodných riešení založená na jazyku Java a distribuovaná pod GNU Lesser General Public Licence. Medzi základne vlastnosti Liferay portálu patria:

- nezávislosť na platforme,
- viacjazyčnosť užívateľského rozhrania,
- zabezpečená SSO autentizácia,
- personalizácia webových stránok,
- jednoduché usporiadanie prvkov pomocou drag-and-drop.

Liferay ponúka podporu pre rôzne rozšírenia či už vzhľadu alebo funkčnosti a to rôznymi typmi pluginov: Themes, Layout Templates, Portlet. Portlet je samostatná aplikácia napísaná v jazyku Java, ktorá tvorí vlastný obsah portálu.

4 Aplikácia Raynet Connector

4.1 O aplikácií

Počas praxe som spolupracoval na tvorbe webovej aplikácie s názvom Raynet Connector. Táto aplikácia slúži zákazníkom na import faktúr vygenerovaných v XML podobe z ekonomického systému POHODA do systému Raynet CRM. Každý užívateľ sa do aplikácie prihlasuje pod vlastným užívateľským meno a heslom. Má k dispozícií viaceré užívateľské role. Admin nastavuje užívateľom rôzne nastavenia a práva obmedzujúce funkcie pre užívateľa. Každý užívateľ musí mať meno, inštanciu a API kľúč, čo sú potrebné prihlasovacie údaje do systému Raynet CRM. Ďalej si užívateľ môže nastavovať vlastné nastavenia, ktoré sa týkajú samotného importovania faktúr ako napríklad: nastavenie vlastníka obchodného prípadu, spôsob importovania nových produktov, nastavenia kategórie obchodných prípadov a iné. Pre každý prevedení import má užívateľ k dispozícií log o danom importe.

Všetky podrobnosti o projekte a úlohy, ktoré som dostal zadané boli vo forme priameho kontaktu s vedúcim riaditeľom vývoja alebo cez webovú stránku na zadávanie úloh k projektom Redmine. Každá zadaná úloha bola konzultovaná s vedúcim vývoja. Celý projekt, na ktorom som pracoval bol vytvorený od začiatku až po samotnú fungujúcu aplikáciu.

4.2 Príprava

Na začiatku praxe bolo potrebné, aby som mohol začať pracovať, nainštalovať všetky potrebné nástroje, ktoré boli nutné pre vývoj konkrétnej aplikácie. Najskôr som si musel stiahnuť a nainštalovať IDE. To mi bolo odporúčané Eclipse Java EE IDE for Web Developers. Nakoľko som už mal menšiu skúsenosť s Eclipe Java, zorientovanie sa v ňom mi to uľahčilo. Ďalej som potreboval MySQL Wokrbench, čo je vizuálny nástroj na správu databázy. Okrem iného som potreboval aj webový server, ten sme zvolili Apache Tomcat kvôli jeho jednoduchosti a bezplatnosti.

Nasledujúcou úlohou bolo nainštalovanie a sprevádzkovanie nástroja GIT. Firma používala nástroj GitLab - webový repozitár ktorý umožňuje systémovu správu verzií. Tento nástroj bol pre mňa celkom nový tak na začiatku mi trvalo dlhšie kým som sa naučil používať základne úkony s gitom a to operácie:

- commit-uloženie zmien do lokálného repozitára,
- pull-sťahovanie dát zo vzdialeného repozitára,
- push-odosielanie lokálnych záznamov do vzdialeného repozitára.

Po inštalácií a konfigurácií všetkého žiadúceho pre prácu nám bol na GIT vedúcim nahraný cvičný projekt, na ktorom sme si mali vyskúšať prácu s Javou a frameworkom PrimeFaces, s ktorým sme potom neskôr počas celej praxe pracovali. Počas prípravy ako aj počas celej praxe, som si pozeral rôzne tutoriály, príklady a študoval som danú problematiku.

4.3 Užívatelia

Aplikácia beží na serveri Tomcat a prístup k nej majú viacerí užívatelia, preto bolo potrebné v aplikácií zabezpečiť správu týchto užívateľov a ich ukladanie pretože, každý užívateľ ma vlastné nastavenia importu, osobné nastavenia, rôzne práva a role. Pre užívateľa bola vytvorená trieda User. Táto trieda sa pomocou frameworku Hibernate mapuje na entitu ktorá sa ukladá do databázy aplikácie.

4.3.1 Role užívateľov

Každý užívateľ môže mať nastavené viaceré role, v závislosti od typu užívateľa. Boli mi zadané tri užívateľské role: Admin, Config, User. Pre ukladanie roli samotných som vytvoril triedu Role ktorá sa pomocou hibernate mapovala do databázy. Ďalej interface IRoleDao, v ktorom sa definovali metódy na prácu s databázou (CRUD) operácie a tie boli implementované v RoleDAOHibernate. Pre priradenie jednotlivých roli užívateľom som vytvoril triedu UserRoles artribútmi User a Role, v databáze sa potom jednotlivé záznamy skladali z ID role a ID užívateľa. Na výpise 1 vidíme ukážku riešenia kardinality vzťahov entít v tabuľke UserRole pre entitu Role.

```
/**
     * @return the role
     */
    @ManyToOne
    @JoinColumn(name = "role")
    public Role getRole() {
        return this.role;
    }
```

Výpis 1: Ukážka riešenia kardinality vzťahov entít

Podobne ako pre už pre spomínajúce Role mala svoj intreface a UserRoleDAOHibernate, v ktorom boli definované metódy typu vráť všetky role používateľa, zisti či ma daný užívateľ rolu a iné. Pri implementácií ako užívateľov aj iných funkcií sme používali návrhový vzor DAO Factory. Na výpise 2 vidíme ukážku metódy, pomocou ktorej dostaneme z databázy Uid rolí daného užívateľa, ktory sa vkladá ako paramater.

```
@Override
    public List<String> getRoleUids(String user) throws DAOException {
        List<String> result = Collections.emptyList();
        Session ses = SessionUtil.getCurrentSession();
        try {
            result = ses
                    .createQuery("select distinct ur.role.uid from " + UserRole.
                        class.getName() //$NON-NLS-1$
```

```
                        + " ur where ur.user.username=:name", String.class)
                            //$NON-NLS-1$
                    .setParameter("name", user).getResultList(); //$NON-NLS-1$
        } catch (HibernateException he) {
            throw new DAOException(he.getMessage());
        } finally {
            ses.close();
        }
        return result;
    }
```

Výpis 2: Ukážka metódy obsluhujúca databázu

4.3.2 Zobrazovanie užívateľov

Ako frontend v celej aplikácií bol použitý framework PrimeFaces. Pre zobrazenie používateľov a prácu s nimi som potreboval dva Java Beany (programová komponenta, pomocou ktorej sa dá vizuálne pracovať). Vytvoril som si preto UsersViewBean a UserDetailViewBean.

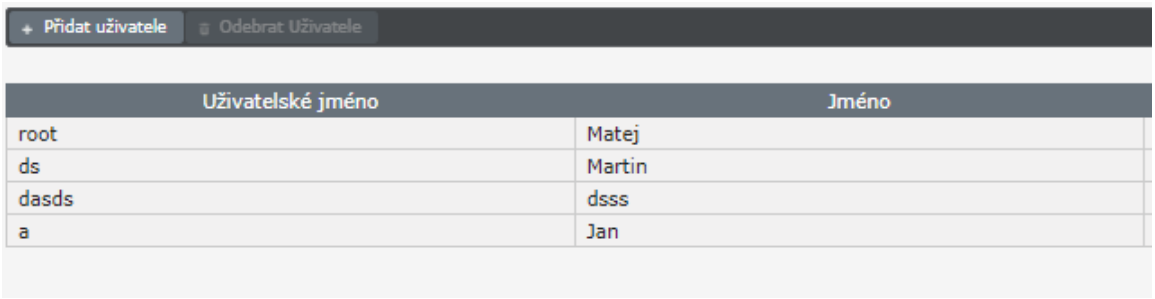
- UsersViewBean obsahoval metódy ako: pridanie/odobranie nového užívateľa, vrátenie zoznamu užívateľov a podobné.
- UserDetailViewBean obsahoval metódy na prácu už s vybraním užívateľom ako: zmena hesla a osobných údajov, zmena nastavení a podobné.

Následne som vytvoril potrebné XHTML súbory, v ktorých sa implementovala už samotná vizualizácia stránky s užívateľmi. Napríklad pre zobrazenie zoznamu užívateľov som využil komponentu frameworku PrimeFaces dataTable. Vo Výpise 3 vidíme ukážku komponenty Primefaces: dataTable. Tabuľka je naplnená pomocou metódy getusersList z Beanu usersViewBean.

```
<p:dataTable
        id="users-table"
        var="user"
        value="#{usersView.usersList}"
        selectionMode="single"
        selection="#{usersView.selected}"
        emptyMessage="#{i18n['table-empty-message']}"
        rowKey="#{user.username}">
        <p:ajax
            event="rowSelect"
            update=":users-form:toolbar" />
        <p:column headerText="#{i18n['user-username']}">
```

```
            <h:outputText value="#{user.username}" />
        </p:column>
```

Výpis 3: Ukážka xhtml súboru pre zobrazenie uživateľov



Obr. 1: Ukážka zobrazovania užívateľa

4.3.3 Logovanie akcií užívateľov

Pre prehliadnejšiú správu užívateľov, som iplmentoval funkciu logovanie akcií užívateľov. Táto funkcia zaznamenáva všetky akcie ktoré, každý užívateľ v aplikacií vykoná. Napríklad: prihlásenie a odhlásenie užívateľa, zmena údajov o užívateľovi, pridanie a odobranie modulu užívateľovi, nahranie súboru užívateľom, prevedenie importu uživateľom, zmena nastavení importu. Zobrazovanie stránky logovanie akcií je dostupné iba pre používateľov, ktorí majú nastavenú uživateľskú rolu Admin. Pre ukladanie akcií uživateľov som si vytvoril v databáze tabuľku logs, ktorá obsahovala atribúty:

- user-pre zaznamenia ID užívateľa, ktorý danú akciu uskutočnil,
- create- dátum uskutočnenia akcie,
- message-informácia o danej akcií,
- type- typ danej akcie, napríklad: login, logout, user setting add.

Následne rovnako ako pri uživateľských rolách som vytvoril interface IlogsDao s abstraktnými metódami na práu s tabuľku logs v databáze. Potom som vytvoril triedu LogsDAOHibernate ktorá implementovala tento interface. Ďalej mi stačilo už, iba implementovat konkretný Bean pre logy viď. Výpis4 a XHTML stránky pre vizualizáciu.

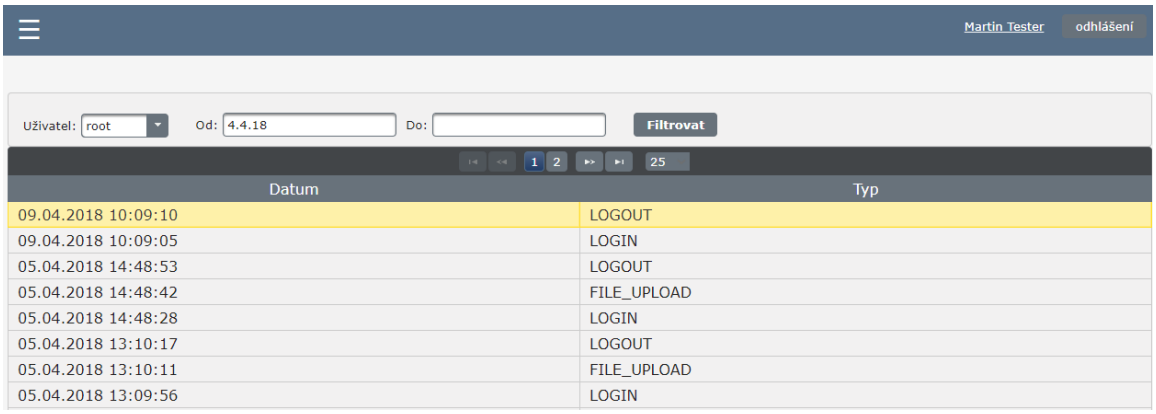
```
public List<Logs> getLogs() {
        if (this.logs == null) {
            try {
                Calendar c;
                c = new GregorianCalendar();
```

```
            if (this.to != null) {
                c.setTime(this.to);
                c.add(Calendar.DAY_OF_MONTH, 1);
            }
            this.logs = DAOFactory.getInstance().getLogsDAO().
                getByUserAndDates(this.selectedUser, this.from, c.getTime());
        } catch (DAOException e) {
            LOG.error(e.getMessage(), e);
        }
    }
    return this.logs;
}
```

Výpis 4: Ukážka metódy getLogs() v LogsBean

Vo výpise 4 môžme vidieť metódu getLogs() ktorá nám vracia v štruktúre List zoznam logov. Logy sa filtrujú pomocou metódy getByUserAndDates, ktorej parametre zadáva užívateľ už na zobrazenej stránke viď. Obr.2.

Užívateľ si vyberie užívateľa ktorého logy akcií chce zobraziť a následne rozmedzie dátumu od a do kedy boli jednotlivé akcie prevedené. Parametre: uživateľ a dátum od sú povinné, pokiaľ užívateľ nezadá parameter do, tak ten sa automatický nastaví na aktuálny dátum. Po kliknutí na konkrétny riadok v tabuľke logov, sa zobrazí podrobná správa o prevedenej akcii.



Obr. 2: Ukážka zobrazovania logov akcií

4.4 Web Services

Pre správne fungovanie importu faktúr sme potrebovali pracovať s dátami vnútri samotného CRM systému. Na toto sme použili cloud CMR REST API. Je to programové rozhranie systému RAYNET CRM určené pre aplikácie tretích strán. Komunikácia prebieha štandardným HTTP protokolom. Toto API sme využívali napríklad na získanie zoznamu firiem, obchodných

prípadov a ich kategórií, zoznamu osôb, produktov, ich detaily, zakladania nových produktov a veľa rôznych iných potrebných údajov ktoré sme potrebovali pre správnu funkčnosť importu faktúr. Všetky dáta sa prenášali v štruktúre JSON.

```
{
    "success": "true",
    "totalCount": 2,
    "data": [
        {
            "id": 3,
            "titleBefore": "Ing.",
            "firstName": "Petr",
            "lastName": "Novák",
            "titleAfter": "CSc.",
            "primaryRelationship": {
                 "id": 1,
                 "type": "konzultant",
                 "company": {
                   "id": 1,
                   "name": "T.S.
```

Výpis 5: Ukážka tela JSONU

4.4.1 Osoby API

Dostal som za úlohu získať zoznam osôb z CRM systému. O osobách som mal získavať ich id, titul, meno a priezvisko. Vytvoril som triedu Person s atribútmi id, title, firstName a lastName. Potom triedu PersonResponse ktorá obsahovala dátovú štruktúru List do ktorej sa ukladajú osoby získane z CRM.

Samotné získavanie prebiehalo v triede PersonWS ktorá dedila z predpripravenej abstraktnej triedy. Táto abstraktná trieda obsahovala metódy ako GET, POST, PUT, ktoré podľa požiadavky spracovávala daný JSON. Potom stačilo už len v PersonWS implementovať metódu getPerson v ktorej sa vytvoril objekt PersonResponse a bol naplnený pomocou už spomínanej metódy get kde sa ako jej parameter sa nastavilo už konkrétne API z Raynetu CRM.

Obdobným spôsobom sme získavali aj všetky ostatné potrebné údaje.

```
public List<Person> getPerson(String name) throws CommunicationException {
        PersonResponse pr = get("https://app.raynet.cz/api/v2/person/?
            userAccount-id[NE]", Collections.emptyMap());
        if (pr != null) {
            LOG.debug(pr);
```

```
        if ((pr.getData() != null) && !pr.getData().isEmpty()) {
            return pr.getData(); }
        }
    return null;...
```

Výpis 6: Ukážka kódu metódy getPerson()

4.5 Import

Počas praxe bola aplikácia schopná načítaváť a importovať XML faktúry z účtovacieho systému POHODA do systému Raynet CRM. Pri implementácií sa však dbalo aj na to že v budúcnosti bude následne implementované, načítanie faktúr aj z iných účtovacích systémov.

Pri importovaní užívateľ najskôr nahrá súbor. Následne sa súbor kontroluje či ma požadovaný formát. V prípade chybného formátu alebo štruktúry sa vypíše hláška nepodporovaného formátu. Po úspešnom nahraní súboru sa užívateľovi zobrazilo menu s načítanými faktúrami. V tomto menu rovnako ako aj v osobných nastaveniach si mohol meniť jednotlivé nastavenia súvisiace s importom. Nastavenia klientov, produktov a obchodných prípadov. Po prebehnutom importe bola stránka presmerovaná na stránku s podrobnými informáciami o prevedenom importe. Samotný import obsahoval rôzne pravidlá ako sa majú faktúry, produkty, obchodné prípady spracovávať.

Napríklad pokiaľ obchodní prípad neexistoval založil sa nový a vytvorili sa väzby na príslušné produkty. Pokiaľ existoval a obsahoval iné produkty ako v importovanej faktúre, prebehla oprava na základe faktúry.

5 Liferay Portal

Na konci mojej praxe som sa zoznámil aj s prostredím Liferay Portal. Pomocou Gitu nám bol importovaný projekt na ktorom som si vyskúšal prácu v tomto prostredí. Súčasťou Liefray portálu su jednotlivé portlety, ktoré môžeme pomocou Liferay CMS (systém pre správu obsahu) pridávať a odoberať na jednotlivé stránky. Zásadna zmena pri práci s Liferay bola práca s databázou. Pri samotnej inštlácií prostredia Liferay Portal ponúkne sám integrovanie s databázou. V mojom prípade som využil MySQL. Ak potrebujeme mapovať nejakú entitú do databázy využívame na to Liferay Service Builder. Po pridání nového Liferay Service Builder sa nám vytvorí súbor service.xml. Do tohto súboru už zadávame a definujeme pomocou tagov jednotlivé entity viď. Výpis7. Pri každej entite ktorú vytvoríme Liferay ponúka local-service a remote-service. Na moje účely mi postačovalo local- service.

- Local-service služba vygeneruje lokálne rozhranie služby.
- Remote-service služba vygeneruje vzdialené rozhranie služby napríklad pre mobilného klienta alebo Simple Object Access Protocol.

```
<entity name="MemberInfo" local-service="true" remote-service="false">
      <column name="id" type="long" primary="true"></column>
      <column name="member" type="long"></column>
      <column name="modifiedDate" type="Date"></column>
      <column name="value" type="String"></column>
      <column name="description" type="String"></column>
      <column name="infoType" type="long"></column>
```

Výpis 7: Ukážka entity MemberInfo

Po vytvorení entity v service.xml sa spustí funkcia service builder ktorá vytvorí rozhranie služby. Automaticky vygeneruje kód k implementacií základných metód viď. Výpis8, ktoré sú schopné pracovať s databázou, tieto metódy môžme následne pridávať a editovať.

```
public MemberInfo updateMemberInfo(
      cz.vrk.pi.crm.model.model.MemberInfo memberInfo)
      throws com.liferay.portal.kernel.exception.SystemException {
      Object returnObj = null;

      try {
          returnObj = _invokableLocalService.invokeMethod(_methodName15,
                  _methodParameterTypes15,
                  new Object[] { ClpSerializer.translateInput(memberInfo) });
      } catch (Throwable t) {
```

```
            t = ClpSerializer.translateThrowable(t);
```

Výpis 8: Ukážka automaticky vygenerovanej metódy updateMemberInfo

Takáto vygenerována metóda sa môže potom volať pomocou:
MemberInfoLocalServiceUtil.updateMemberInfo().

6 Znalosti uplatnené v priebehu praxe

Počas mojej praxe som uplatnil mnohé znalosti získané počas štúdia. Ako základne znalosti objektovo orientovaného programovania z predmetov Programováni II, Alogritmy I a II. Taktiež som využil znalosť jazyku Java z predmetu Programoací jazyky I. Pri implementácií väčšie aplikácie som uplatnil znalosti získane v predmetoch Vývoj informačních systémů a Úvod do softwarového inženýrství. Pri práci s databázou mi boli cenné skúsenosti z predmetov Úvod do databázových systémů a Databázové a informační systémy.

7 Znalosti chýbajúce v priebehu praxe

Počas praxe som sa stretol s chýbajúcimi znalosťami systému Git, frameworkom PrimeFaces a Hibernate. Počas praxe som tieto znalosti musel neustále doplňovať samoštúdiom

8 Výsledky absolvovanej praxe a jej zhodnotenie

Výsledkom mojej práce bola aplikácia Raynet Connector, ktorú už v súčasnosti používajú zákazníci. Absolvovanie odbornej praxe hodnotím ako vynikajúcu skúsenosť a som veľmi vďačný, že som mohol absolvovať takúto formu bakalárskej práce. Počas praxe som získal mnoho skúseností v oblasti jazyku Java a vývoja webových aplikácií. To hodnotím ako veľké pozitívum, pretože ako študentovi mi skúsenosti z praxe, sú obrovským prínosom.

9 Záver

Hlavným cieľom tejto práce bolo popísať činnosť ktorú som vykonával na 50 dennej odbornej praxi vo firme SCOVECO s.r.o. V tejto práci som sa venoval hlavne projektu Raynet Connector na ktorom som spolupracoval v tíme.

Na začiatku praxe mi chvíľu trvalo kým sa začal orientovať v takom veľkom projekte, pretože počas štúdia som sa s ničím takým nestretol. Neskôr však po získaní skúseností som vedel efektívnejšie pracovať. Absolvovanie odbornej praxe bola pre mňa cenná skúsenosť. Zúčastniť sa na reálnom projekte a získať skúsenosti z praxe sú neoceniteľnou výhodou pre každého študenta.

Literatura

[1] GSON [online] [cit2018-04-20].Dostupné z WWWW: https://www.json.org/json-cz.html

[2] GIT [online] [cit2018-04-20].Dostupné z WWWW: https://git-scm.com/doc

[3] Hibernate [online] [cit2018-04-20].Dostupné z WWWW: https://www.tutorialspoint.com/hibernate/index.htm

[4] JSF [online]. [cit2018-04-20]. Dostupné z WWWW: http://www.oracle.com/technetwork/topics/index-090910.htmll

[5] Java [online]. [cit2018-04-20]. Dostupné z WWWW: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

[6] Eclipse [online]. [cit2018-04-20]. Dostupné z WWWW: https://www.tutorialspoint.com/eclipse/index.htm

[7] Liferay Portal [online]. [cit2018-04-20]. Dostupné z WWWW: https://github.com/liferay/liferay-portal

[8] PrimeFaces [online]. [cit2018-04-20]. Dostupne z WWWW: https://www.primefaces.org/showcase/

[9] Shiro [online]. [cit.2018-20-04]. Dostupne z WWWW: https://shiro.apache.org

[10] Tomcat [online]. 2015 [cit.2018-20-04]. Dostupne z WWWW: http://tomcat.apache.org